|  |  |
| --- | --- |
| Application can fetch the data from the internal API and store in properly (upon init) | Приложение может получать данные из внутреннего API и сохранять их должным образом (при инициализации) |
| Application can provide you with search results (full-text search, based on all the available fields, including tags) | Приложение может предоставить вам результаты поиска (полнотекстовый поиск по всем доступным полям, включая теги) |
| The search term is applied to any part of the field, not only the beginning | Поисковый запрос применяется к любой части поля, а не только к началу |
| The picture info re-fetched and re-cache every X seconds. X is configurable | Информация об изображении повторно выбирается и повторно кэшируется каждые X секунд. X настраивается |
| Clean and readable code:  -understandable naming;  -Logic is clear and can be simply read from the code;  -Code follows to any kind of formatting standard;  - no spaghetti code, when object is created on one level and is populated deeply in child calls;  -methods/functions are short and don't have mixed reponsibility levels;  -cade is clean; it doesn't have commented out code and non-used methods; if/for without bracers;  -correct incapsulation;  Ideal solution is 20 points. Each issue subtracts -5 or so; it means that candidate can go to negative value, but 0 is minimum here. | Чистый и читаемый код: - понятное нейминг; -Логика понятна и легко читается из кода; -Code следует любому стандарту форматирования; - отсутствие спагетти-кода, когда объект создается на одном уровне и глубоко заполняется дочерними вызовами; -методы / функции короткие и не имеют смешанных уровней ответственности; -корпус чистый; в нем нет закомментированного кода и неиспользуемых методов; если / без наручей; -правильная инкапсуляция;  Идеальное решение - 20 баллов. Каждый выпуск вычитает -5 или около того; это означает, что кандидат может перейти в отрицательное значение, но здесь минимум 0. |
| Search analysis algorithm is flexible and supportable.  -Algorithm logic is clear and understandable.  -Algorithm is well-structered and supportable.  -No excessive classes that adds no value (no over-engineering).  Ideal sulution is 15 points. Each issue subtracts -5 or so; it means that candidate can go to negative value, but 0 is minimum here. | Алгоритм поискового анализа гибкий и поддерживаемый. -Алгоритм логики понятен и понятен. -Алгоритм хорошо структурирован и поддерживается. -Нет чрезмерных занятий, которые не добавляют ценности (без чрезмерной инженерии). Идеальное решение - 15 баллов. Каждый выпуск вычитает -5 или около того; это означает, что кандидат может перейти в отрицательное значение, но здесь минимум 0. |
| Executable file is provided and runs. Readme exists. Output of application work results is attached.  Logging is applied through classes  Search is scalable, there's any sort of partial result caching or other search optimization - OPTIONAL  Application logs statistical information that is used for dicision making - OPTIONAL  Unit or integration test - OPTIONAL | Исполняемый файл предоставляется и запускается. Readme существует. Вывод результатов работы приложения прилагается. Ведение журнала применяется через классы Поиск масштабируемый, есть какое-либо частичное кеширование результатов или другая поисковая оптимизация - НЕОБЯЗАТЕЛЬНО Приложение регистрирует статистическую информацию, которая используется для принятия решений - НЕОБЯЗАТЕЛЬНО Модульный или интеграционный тест - ДОПОЛНИТЕЛЬНО |
|  |  |
|  |  |
|  |  |
|  |  |

|  |  |
| --- | --- |
| **Intro**  Imagine that you are involved in the development of a large file storage system. Special feature here is storing photos and images. We need to provide our users with the possibility to search stored images based on attribute fields.  **Requirements**   1. We need to see your own code. 2. The app should load and cache photos from our API endpoint http://interview.agileengine.com 3. Obtain a valid Bearer token with valid API key (don't forget to implement invalid token handler and renewal)  POST http://interview.agileengine.com/auth  Body: { "apiKey": "23567b218376f79d9415" }  Response: { "token": "ce09287c97bf310284be3c97619158cfed026004" } 4. The app should fetch paginated photo feed in JSON format with the following REST API call (GET):  GET /images  Headers: Authorization: Bearer ce09287c97bf310284be3c97619158cfed026004  Following pages can be retrieved by appending ‘page=N’ parameter:  GET /images?page=2  No redundant REST API calls should be triggered by the app. 5. The app should fetch more photo details (photographer name, better resolution, hashtags) by the following REST API call (GET): GET /images/${id} 6. The app should fetch the entire load of images information upon initialization and perform cache reload once in a defined (configurable) period of time. 7. The app should provide a new endpoint: GET /search/${searchTerm}, that will return all the photos with any of the meta fields (author, camera, tags, etc) matching the search term. The info should be fetched from the local cache, not the external API. 8. You are free to choose the way you maintain local cache (any implementation of the cache, DB, etc). The search algorithm, however, should be implemented by you. 9. We value code readability and consistency, and usage of modern community best practices and architectural approaches, as well, as functionality correctness. So pay attention to code quality. 10. Target completion time is about 2 hours. We would rather see what you were able to do in 2 hours than a full-blown algorithm you’ve spent days implementing. Note that in addition to quality, time used is also factored into scoring the task.   **Expected Deliverables**   1. Source code. 2. Readme, with instructions, how to build and run. | вступление  Представьте, что вы участвуете в разработке большой системы хранения файлов. Особенностью здесь является хранение фотографий и изображений. Нам необходимо предоставить нашим пользователям возможность поиска сохраненных изображений на основе полей атрибутов.  Требования  1-Нам нужно увидеть ваш собственный код. 2-Приложение должно загружать и кэшировать фотографии с нашей конечной точки API http://interview.agileengine.com.  3-Получите действительный токен на предъявителя с действующим ключом API (не забудьте реализовать обработчик недопустимого токена и продление) ЗАПИСЬ http://interview.agileengine.com/auth Тело: {"apiKey": "23567b218376f79d9415"} Ответ: {"токен": "ce09287c97bf310284be3c97619158cfed026004"}  4-Приложение должно получить фид фотографий с разбивкой на страницы в формате JSON с помощью следующего вызова REST API (GET): GET / изображения Заголовки: Авторизация: предъявитель ce09287c97bf310284be3c97619158cfed026004 Следующие страницы можно получить, добавив параметр page = N: GET / images? Page = 2 Приложение не должно запускать избыточные вызовы REST API.  5-Приложение должно получить дополнительные сведения о фотографии (имя фотографа, лучшее разрешение, хэштеги) с помощью следующего вызова REST API (GET): GET / images / $ {id}  6-Приложение должно получить всю информацию об изображениях при инициализации и выполнить перезагрузку кеша один раз в определенный (настраиваемый) период времени.  7-Приложение должно предоставить новую конечную точку: GET / search / $ {searchTerm}, которая вернет все фотографии с любым из мета-полей (автор, камера, теги и т. Д.), Соответствующих поисковому запросу. Информация должна быть получена из локального кеша, а не из внешнего API.  8-Вы можете выбрать способ поддержки локального кеша (любая реализация кеша, БД и т. Д.). Однако алгоритм поиска должен быть реализован вами.  9-Мы ценим читабельность и согласованность кода, использование передовых практик и архитектурных подходов современного сообщества, а также правильность функциональности. Так что обратите внимание на качество кода.  10- Целевое время выполнения около 2 часов. Мы предпочли бы увидеть, что вы смогли сделать за 2 часа, чем полноценный алгоритм, на реализацию которого вы потратили дни.  Обратите внимание, что помимо качества при подсчете баллов за задачу также учитывается затраченное время.  Ожидаемые результаты  Исходный код.  Readme, с инструкциями, как собрать и запустить. |